Formulaire ADO

Sommaire

I. *Le codage de l'information*

1. Rappel sur les bases

- Binaire -> Décimal

$(A)_2 = a_0 a_1 a_2 \dots a_n \Rightarrow (A)_{10} = \sum a_i * 2^i$.

- Décimal -> Binaire

Partie entière	Partie fractionnaire
	↓
34/2 = 17 r = **0**	0.625 *2 = **1**.25
17/2 = 8 r = **1**	0.25*2 = **0**.5
8/2 = 4 r = **0**	0.5*2 = **1**
4/2 = 2 r = **0**	
2/2 = 1 r = **0**	
1/2 = 0 r = **1**	
↑	

$$(34.625)_{10} = (100010.101)_2$$

- Binaire <-> Hexadécimal

On regroupe les octets par 4. Un groupe de 4 octets correspond à un chiffre hexadécimal.
A l'inverse, un chiffre hexadécimal est remplacé par un groupe de 4 octets, le groupe codant la valeur du chiffre hexadécimal.

2. Codage des nombres sur un ordinateur

- Les entiers relatifs : le complément à 2

Pour coder un nombre négatif, on prend sa représentation entière en valeur absolue, on inverse tous les bits, et on ajoute 1.

- Les flottants : la norme IEEE 754

On considère que nous avons 32 ou 64 bits selon la précision à notre disposition. On alloue un bit au signe, une partie des bites afin de décrire l'exposant et le reste à la représentation du nombre normalisé :

	Signe	Exposant	Mantisse
Simple Précision	1 bit	8 bits	23 bits
Double Précision	1 bit	11 bits	52 bits

Ainsi un nombre normalisé vaut : $(-1)^{Signe} * 1, Mantisse * 2^{Exposant-Biais}$.
Le biais se calcul comme suit : $2^{Exposant-1} - 1$. Ainsi en simple précision il vaut 127 et en double précision 1023.

Quelques cas particuliers :

Signe	Exposant	Mantisse	Valeur
-	0 … 0	0 … 0	0
-	1 … 1	0 … 0	$\pm\infty$
-	1 … 1	-	NaN
-	0 … 0	-	Nb dénormalisé

3. Correction de codes

Plusieurs solutions existent, dont certaines ont été vues en théorie de l'information :

- Contrôle de parité => détecte un nombre impair d'erreurs, mais pas de correction
- Double parité => détecte un nombre impair d'erreurs et permet de corriger
- VRC(vertical redundancy check)/LRC(longitudinal redundancy check) => 0 si pair 1 si impair
- Distance de Hamming => ajout de k bits de test aux m bits de donnée. Lorsque l'on détecte une erreur, on corrige le mot erroné par le mot valide qui lui est le plus proche (en terme de nombre de différences de bit)

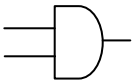
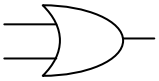
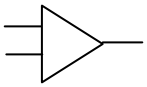
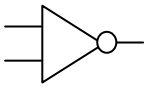
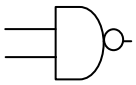
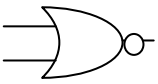
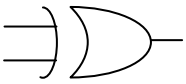
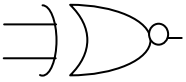
II. *Un peu d'électronique*

1. Rappels d'électronique

Circuit combinatoire : circuit dont la fonction de sortie s'exprime par une expression logique de ses seules entrées.

Circuit séquentiel : circuit dont la fonction de sortie dépend non seulement de ses entrées mais aussi de l'état antérieur du circuit.

Rappels de composants électroniques :

OPERATEUR	SYMBOLE
AND : $a.b$	
OR : $a + b$	
BUFFER : a	
NOT : $\bar{a}$	
NAND : $\overline{a.b}$	
NOR : $\overline{a + b}$	
XOR : $a \oplus b$	
XNOR : $\overline{a \oplus b}$	

2. Architecture de Von Neumann

Une machine de Von Neumann est caractérisée par :

- Une mémoire séquentielle (programme, données, organisation en vecteur de mots)
- Une unité de calcul arithmétique ou logique : AOL
- Une unité d'entrées-sorties (échange d'informations)
- Une unité de commande (chef d'orchestre)

Fonctionnement d'une machine de Von Neumann :

- Extraction de l'instruction (depuis la mémoire)
- Analyse de l'instruction
- Recherche des données impactées par l'instruction
- Déclenchement de l'opération sur l'unité de calcul (ou unité d'entrée/sorties)
- Stockage du résultat dans la mémoire

Pour permettre le fonctionnement décrit ci-dessus, une machine de Von Neumann a besoin en plus :

- D'une horloge qui permet de synchroniser les différents dispositifs
- De registres qui mémorisent les informations binaires (chargement soit sur le niveau de l'horloge soit sur le front de l'horloge)
- De bus qui permettent le transit de l'information d'une unité à l'autre

Une mémoire est un tableau (ensemble de cellules). L'adresse d'une donnée correspond à sa position (index) dans la mémoire. Dans une mémoire, on ne peut faire que deux opérations : lire et écrire.
Pour accéder à une donnée, on utilise un registre d'adresse (RA). Pour effectuer une lecture ou écriture, on passe par le registre d'échange ou registre mot (RM).

Voici les instructions pour effectuer une lecture (pas de perte d'information) :

1. RAD <- adresse
2. RM <- Mémoire[RAD]

Voici les instructions pour effectuer une lecture (perte d'information) :

1. RAD <- adresse
2. RM <- valeur
3. Mémoire[RAD] <- RM

3. Unité Arithmétique et Logique

L'UAL peut se voir comme une fonction composée de trois paramètres. Le premier paramètre est le code de l'opération à effectuer, et les deux suivants sont les arguments nécessaires à l'exécution de l'opération. En général, une opération retourne une valeur résultat.
Dans la majorité des cas, les arguments proviennent de la mémoire, et le résultat est stocké dans la mémoire.

4. Unité d'entrées-sorties

L'unité d'entrées-sorties est une interface qui permet l'échange d'informations (dans un sens ou dans les deux sens selon les périphériques) entre l'unité central et les périphériques. L'unité est composée d'un registre de sélection de périphérique (RS) qui mémorise l'adresse du périphérique et un registre d'échange (RE), équivalent du registre mot RM, qui sert d'interface.

Voici les instructions pour récupérer des données depuis un périphérique :

1. RS <- adresse du périphérique
2. RM <- Périphérique[RS]

5. Unité de commande

L'unité de commande est un dispositif qui permet de récupérer une instruction depuis la mémoire puis de l'analyser. Pour ce faire nous avons besoin de deux nouveaux registres : le compteur ordinal ou compteur de programme (PC) qui contient en début de cycle l'adresse de l'instruction, et le registre d'instruction (RI) qui mémorise l'instruction.

III. Les mémoires

1. Généralités

La finalité d'une mémoire est d'enregistrer une donnée, de la conservée puis de la restituer lorsque nous en avons besoin. Ainsi, l'efficacité d'une mémoire est évaluée à partir de plusieurs critères : la capacité à mémoriser (volume), la capacité à conserver (volatilité) et la capacité à restituer (vitesse).
Un mot mémoire est un ensemble de bits pouvant être lus écrits simultanément.

Il existe deux sortes de mémoires qui sont classifiées selon leur capacité de conservation (volatilité) : les temporaires qui ont généralement besoin d'une alimentation électrique (RAM, registres) et les permanentes qui n'ont pas besoin d'être alimentées (CR-DOM, ROM).

Petit rappel sur les capacités de stockage :

- $2^{10} = 1K = 1\,024$
- $2^{20} = 1M = 1\,048\,576$
- $2^{30} = 1G = 1\,073\,741\,824$
- $2^{40} = 1T = 1\,099\,511\,627\,776$
- $2^{50} = 1P = 1\,125\,899\,906\,842\,620$
- $2^{60} = 1E$
- $2^{70} = 1Z$
- $2^{80} = 1Y$

La performance d'une mémoire ou temps d'accès (temps nécessaire à une opération de lecture ou d'écriture) s'évalue en terme de débit (quantité d'information lues/écrites par unité de temps). Le mode d'accès à la mémoire est un facteur essentiel pour améliorer ces performances temporelles.

Etant donné qu'il est impossible de disposer d'un mémoire rapide illimité (coût engendré trop élevé, impossibilité technologique (le temps d'accès augmente avec la capacité)), il a été décidé de combiner différents types de mémoires. Les performances ainsi obtenues sont excellente grâce à la création d'une hiérarchie : les données les plus utilisées doivent être plus rapides d'accès.

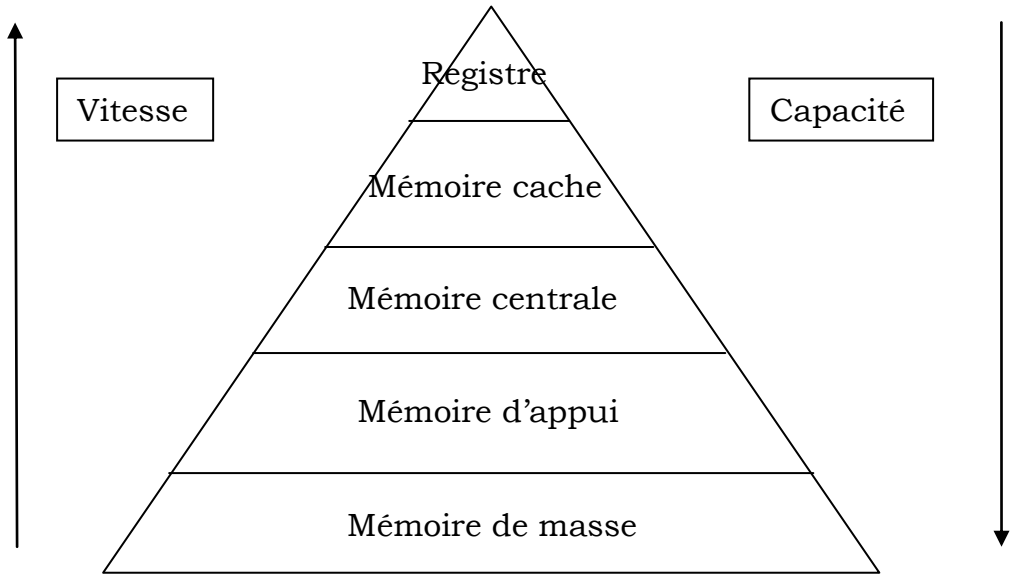


Tableau descriptif des différents types de mémoire

Registre	Cache	Centrale	Appui	Masse
▪ Situé dans le CPU ▪ Grande vitesse ▪ Stockage des opérandes et des résultats ▪ Capacité de 1 mot ▪ Débit : vitesse horloge processeur	▪ Mémoire rapide ▪ Faible capacité (qques Mo) ▪ Rôle de tampon entre CPU et mémoire centrale	▪ Mémoire principale ▪ Temps d'accès élevé ▪ Stockage des données et des programmes ▪ Capacité de qques Go.	▪ Présente dans les systèmes évolués ▪ Intermédiaire entre centrale et masse ▪ Augmente la vitesse d'échange avec les périphériques	▪ Mémoire périphérique ▪ Grande capacité ▪ Coût faible ▪ Lente

2. Mode d'accès

- Accès aléatoire

Mode d'accès le plus utilisé. Une adresse est un mot qui correspond à l'indice de la cellule dans laquelle se trouve la donnée. Ainsi, on désigne de manière fixe et non ambigüe une adresse. Il s'agie du même fonctionnement que celui de la Machine de Von Neumann. Il faut mémoriser l'information à traiter dans le registre d'adresse RA et on utilise le registre de mot RM pour effectuer une lecture/écriture.
A tout instant, n'importe quelle adresse peut être traitée, d'où le nom d'accès aléatoire. La taille d'une adresse dépend de la taille de la mémoire.

- Accès par le contenu

Il s'agit d'une mémoire associative. Ce type d'accès est principalement utilisé dans les mémoires caches. Il s'agit en fait de limiter le nombre de cellules que peut occuper une donnée dans la mémoire cache. Cependant, à une cellule de la mémoire cache correspond plusieurs cellules de la mémoire principale. Ainsi, pour identifier de manière unique une donnée, il y a une notion d'étiquette (qui est une partie de l'adresse de la donnée). Pour plus de clartés, se référer au paragraphe sur la mémoire cache.

- Accès séquentiel

L'accès séquentiel est utilisé pour les mémoires d'archivages (important volume de données). Les opérations écritures et de lectures sont séquentielles. On peut se placer au début de la mémoire (sur la première donnée), lire une donnée (et donc se placer sur la donnée suivante), écrire une donnée et se positionner à la fin (pour ajouter à la fin par exemple). On peut citer l'exemple des bandes magnétiques.

- Accès direct

L'utilisation classique de l'accès direct se fait avec les disques. L'accès se fait bloc par bloc, et une donnée a une position dans un bloc. Les opérations se font comme suit : lecture (bloc, déplacement), écriture (bloc, déplacement, donnée).

- Accès LIFO

Il s'agit d'une pile ; ce type d'accès est essentiellement utilisé dans les piles d'exécutions, car il permet de gérer l'appel et de retour de sous programmes.
Les opérations possibles sont les suivantes : Ecrire (données) ⇔ , Lecture ⇔dépiler, Sommet ⇔ regarder le Sommet sans dépiler, Vide ⇔ teste sur le contenu de la file.

- Accès FIFO

Opérations : Ecriture (donnée) ⇔ ajouter en queue, Lecture ⇔ récupérer en tête, Vide ⇔ teste sur le contenu de la file.

3. Mémoire principale

RAM : stocke des données temporaires. Il en existe de deux sortes :

- RAM dynamique (DRAM) : condensateurs utilisés comme unités de mémorisation ; rafraichissement périodique ; composée d'un transistor et d'un condensateur ;
- RAM statique (SRAM) : bascules utilisées comme unités de mémorisation ; pas de rafraichissement ; plus rapide ; composée de 4 transistors.

ROM : utilisées pour le stockage permanent. Il en existe de 4 sortes :

- ROM : écriture unique lors de la fabrication
- PROM : écriture unique après la fabrication
- EPROM : admet un nombre d'écriture limité
- EAROM : admet un nombre d'écriture illimité

L'élément de base de la mémoire est la cellule. Une cellule possède trois connexions : une entrée (indique si la cellule est activée, c'est-à-dire qu'elle est concernée par l'opération courante), une entrée de contrôle (indique la nature de l'opération courante, à savoir lecture ou écriture) et une ligne bidirectionnelle pour les données (réception ou envoie de la donnée).

IV. La mémoire cache

1. Introduction

Afin d'améliorer les performances des ordinateurs, les constructeurs d'ordinateurs utilisent de plus en plus de mémoires dites cachées. Le principe de ces mémoires cachées a été développé au début des années 1960 par un ingénieur français. Une mémoire cache est une mémoire intermédiaire qui se situe entre un support de données (une autre mémoire dite principale) et le matériel informatique (souvent le microprocesseur) qui souhaite accéder à ces données. Elle est constituée de copies temporaires de données qui proviennent de la mémoire principale. L'intérêt des mémoires caches est leur vitesse d'accès : en effet, du fait de leur proximité physique avec le microprocesseur et de la qualité des matériaux utilisés, les performances d'accès en lecture ou en écriture aux mémoires caches sont grandement améliorées.
On pourrait se poser la question de savoir pourquoi les constructeurs n'utilisent pas uniquement des mémoires de type cache. Deux raisons à ceci ; la première est financière, puisque les coûts des matériaux utilisés pour une mémoire cache sont très élevés. La deuxième raison est d'ordre physique, si on augmente la taille de la mémoire cache, son efficacité en est ralentie puisque l'on a plus de données à rechercher. Toujours dans un souci de performances et pour garantir la plus grande proximité possible entre le microprocesseur et les données, les mémoires caches sont souvent situées dans les microprocesseurs, et la place est donc très limitée ce qui explique la faible capacité de ces mémoires.

L'efficacité des mémoires caches a été confirmée non seulement de manière empirique, mais aussi de manière théorique. En effet, des études sur les comportements des programmes ont permis de tirer deux conclusions :

1. Le principe de localité spatiale

   Lorsque l'on récupère une donnée située à une adresse i, il y a une grande probabilité pour que la prochaine donnée à laquelle on accède soit située à une adresse j qui est physiquement proche de l'adresse d.

2. Le principe de localité temporelle

   Lorsque l'on récupère une donnée à un moment donné t, il y a une grande probabilité pour que l'on ait à nouveau besoin de cette donnée dans un futur immédiat.

2. Fonctionnement

Voici le fonctionnement du processeur en présence d'une mémoire cache :
Le microprocesseur demande une information

- Si cette information se situe dans le cache, celui la renvoie : succès de cache. Sinon, on parle de défaut de cache, et la mémoire cache demande l'information à la mémoire principale
- La mémoire principale renvoie l'information à la mémoire cache
- Une fois enregistrée en mémoire cache (stockée), l'information est transmise au microprocesseur

3. Les différents types de mémoire cache

La mémoire cache étant de petite taille, elle ne peut contenir qu'une petite partie de la mémoire principale. Il faut donc savoir à chaque instant quelle partie de la mémoire principale est stockée dans la mémoire cache. Pour ce faire, on utilise le mapping. Il en existe de trois sortes :

- ***Fully associative cache*** : une donnée quelconque de la mémoire principale peut se situer n'importe où dans la mémoire cache. Efficace au niveau des défauts de cache mais temps de recherche important.
- ***Direct mapped cache*** : la position d'une donnée dans la mémoire cache est entièrement déterminée par sa position dans la mémoire principale. Très efficace pour la recherche, elle est néanmoins très coûteuse en termes de défauts de cache.
- ***N-way set associative cache*** : il s'agit d'un compromis entre les deux versions précédentes, qui assure une bonne efficacité de recherche et un pourcentage de défaut de cache acceptable. C'est cette méthode qui est universellement utilisée de nos jours.

4. Les différentes politiques de remplacement

Lorsque l'on charge une donnée depuis la mémoire principale vers la mémoire cache, il se peut que la mémoire cache soit déjà pleine pleine. Dans ce cas, il faut 'choisir' la donnée de la mémoire cache que l'on va enlever. En accord avec les principes de localité énoncés précédement, il existe plusieurs stratégies de remplacement :

- FIFO : cette méthode consiste à enlever le bloc qui est dans la mémoire depuis le plus longtemps.
- Aléatoire : cette méthode consiste à enlever un bloc de manière alétoire.
- LRU : least recently used. Cette méthode consiste à enlever le bloc qui a été utilisé le moins récemment. Il faut donc ajouter des bits de contrôles (TAG) qui permettent de stocker l'ordre d'utilisation.
- LFU : least frequently used. Cette méthode consiste à enlever le bloc qui a été utilisé le moins fréquement. Il faut donc ajouter des bits de contrôles qui permettent de stocker le nombre d'accès à une donnée.

5. La gestion de l'écriture de données

Lorsque nous voulons écrire dans la mémoire, il existe différentes politiques selon que la donnée se trouve dans le cache ou non.

1. Donnée présente dans le cache

   - Ecrire à la fois dans le bloc du cache et dans le bloc de la mémoire (écriture simultanée / write through)

   - Ecrire uniquement dans le bloc du cache, et différer l'écriture de ce bloc en mémoire principale au moment où il sera 'enlevé' de la mémoire cache, cf politique de remplacement (réécriture / write back). Il faut alors ajouter un TAG activé lorsque l'on a modifié une donnée du cache, afin de ne pas oublier la réécriture en mémoire principale lorsque la donnée est enlevée.

2. Donnée non présente dans le cache

   - Charger le bloc de la mémoire dans le cache puis effectuer l'opération d'écriture (soit write through soit write back) : on parle d'écriture allouée.

   - Effectuer l'écriture directement dans la mémoire principale : on parle d'écriture non allouée.

A partir de ces différentes stratégies, on peut évaluer la performance d'une mémoire en mesurant son temps d'accès moyen.

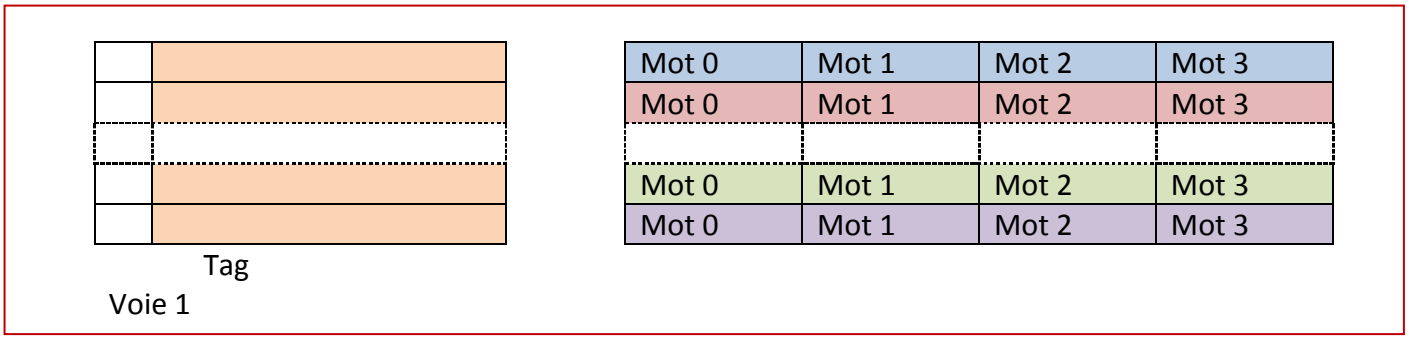
$$temps\ d'accès\ moyen = temps\ d'accès\ succès + taux\ échec * pénalité\ échec$$

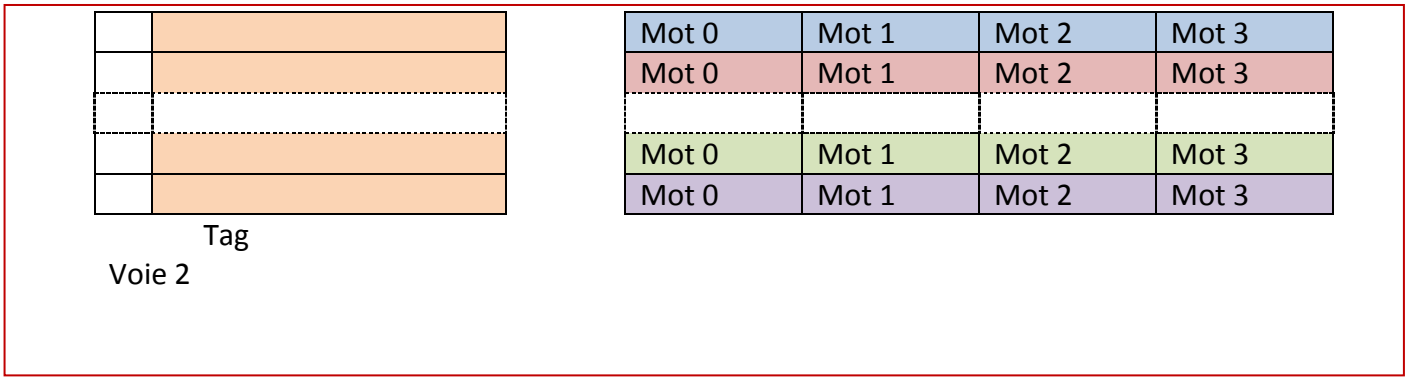
$$temps\ d'accès\ succès = temps\ d'accès à\ une\ donnée\ résidant\ dans\ le\ cache$$

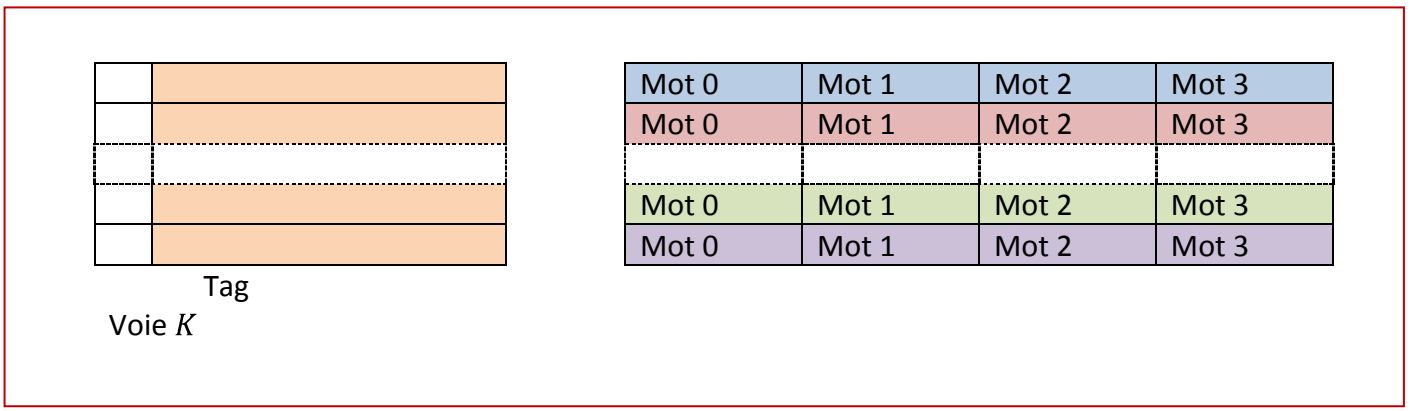
$$taux\ échec = \frac{nbr\ de\ défaut\ de\ cache}{nombre\ d'accès\ au\ cache}$$

6. N-way set associative cache

L'idée est de diviser la mémoire cache en 2^N ensembles distincts. Chaque donnée de la mémoire principale ne peut être chargée que dans un seul ensemble. Cependant, elle peut prendre n'importe quelle place à l'intérieur de cet ensemble. Ainsi, la position d'une donnée dans la mémoire cache étant limité à quelques adresses, la recherche de celle-ci est rapide. Cependant, le fait que la donnée puisse être stockée à plusieurs adresses différentes, limite le nombre de défaut de caches.
L'adresse d'une donnée dans la mémoire cache est divisée en plusieurs partie : il faut identifier la voie de la mémoire cache dans laquelle la donnée se trouve (TAG), repérer l'ensemble considéré (Index), puis récupérer le mot parmi la ligne(Offset).







Dans l'exemple ci-dessus, chaque couleur représente un ensemble. Pour récupérer un mot, il faut connaître la voie dans laquelle il se situe (il y a autant de voies que d'éléments par ensemble), l'ensemble considéré (il y en a 2^N) et enfin le mot parmi la ligne.

V. *L'unité de commande*

1. Petit Rappel

L'unité de commande est un dispositif qui permet d'extraire une instruction depuis la mémoire. Un problème se pose : à quelle adresse va-t-elle lire cette instruction ? Il faut donc un registre particulier qui en début de cycle d'exécution contiendra cette adresse. Il s'agit du compteur ordinal ou compteur de programme (PC).
Ensuite, l'unité de commande analyse l'instruction. Encore une fois, un problème de mémorisation se pose, et nous avons besoin d'un autre registre : le registre d'instruction (RI).

2. Les micro-opérations

Un programme est une suite d'instructions, il faut donc le charger en mémoire. C'est le rôle d'un programme particulier : le chargeur.
L'exécution d'un programme se fait séquentiellement : chacune des instructions sont exécutées les unes après les autres. L'exécution d'une instruction regroupe différentes phases (on parle de cycle d'instruction). Une phase du cycle d'instruction est appelée micro-opération. Une micro-opération est donc une opération fonctionnelle atomique du processeur.
Il est possible dans certains cas de regrouper des micro-opérations afin de les exécuter en parallèle, à condition de respecter la dépendance des données et d'éviter les conflits.

Voici quelques exemples de cycle :

- ***Fetch***
  MAR <- PC
  MBR <- mémoire
  PC <- PC + T
  IR <- MBR
- ***Indirect***
  MAR <- IR(adresse)
  MBR <- mémoire
  IR(aadresse) <- MBR(adresse)
- ***Execute***
  Il est impossible de prévoir cette séquence, elle dépend complètement de l'opcode ; il y a autant de séquences que d'opcodes.
- ***Interrupt***
  MBR <- PC
  MAR <- adresse de sauvegarde
  PC <- adresse de routine d'interruption
  Mémoire <- MBR

3. Le contrôle du processeur

L'unité de commande doit exécuter deux tâches basiques : le séquençage des micro-opérations d'un part, et l'exécution de celles-ci d'autre part. Il existe différentes catégories de micro-opérations :

- Transfert de données entre registres
- Transfert de données d'un registre vers une interface externe
- Transfert de données d'une interface externe vers un registre
- Opération sur l'ALU en utilisant des registres pour les opérandes sources et le résultat

L'UAL interagit avec l'extérieur grâce aux entrées :

- Une micro-opération s'exécute en un top d'horloge.
- Le registre d'instruction, à travers l'opcode qu'il envoie, détermine quelle séquence de micro-opérations doit être exécutée.
- Le registre de condition est utilisé pour déterminer le statut du processeur et les instructions conditionnelles.
- Des signaux de contrôle qui proviennent du bus de contrôle

L'UAL interagit avec l'extérieur grâce aux sorties :

- Des signaux de contrôle interne vers le processeur. Il en existe de deux types : des signaux déclenchant des transferts entre registres et des signaux déclenchant des opérations sur l'ALU
- Des signaux de contrôle transportés par le bus de contrôle. Om en existe de deux type : à destination de la mémoire et à destination des modules d'E/S.

VI. CPU et instructions

1. Les instructions

Lors de la conception d'un CPU, on doit auparavant avoir choisi le jeu d'instruction que le CPU sera capable d'exécuter, c'est-à-dire l'ensemble des instructions reconnues par le CPU.
Nous avons vu dans une partie précédente, que qu'une instruction était en fait un cycle de micro-opérations. Une instruction requiert plusieurs informations : le code de l'opération (opcode), les références aux opérandes (adresses) qui sont selon les cas au nombre de un ou deux, une référence à l'opérande de résultat et une référence à la prochaine instruction.
Le jeu d'instruction doit être suffisamment expressif pour permettre de coder toute instruction d'un langage de haut niveau : en général, à une instruction de haut niveau correspond plusieurs instructions du langage machine.

Lorsque l'on construit un jeu d'instruction, plusieurs aspects doivent être pris en compte :

- Les opérations : combien ? Quelle complexité ?
- Les types de données : quels sont les types supportés par les opérations ?
- Les registres : combien en faut-il ? Comment sont-ils utilisés ?
- L'adressage : quel est le type d'adressage ?
- Le format : qu'elle longueur d'instruction ? En effet, si en théorie il faudrait 4 champs de référence (deux opérandes, résultat et prochaine instruction), il est possible en pratique de réduire le nombre d'adresses. Le choix du nombre d'adresse est un compromis ; si il y a moins d'adresses les instructions sont plus courtes et la CPU plus simple, mais le nombre d'instructions augmente et le temps d'exécution aussi. De plus, plusieurs adresses impliquent autant de registres. En général, on choisit entre 2 et 3 adresses.

S'il existe un très grand nombre d'instructions, il est possible de les catégoriser. En effet, on distingue les instructions :

- De transfert de données (emplacement, taille, moyen d'accès)
- De transfert de contrôle
- Les opérations arithmétiques
- Les opérations logiques
- De transfert de données avec les E/S

Comme nous l'avons ci-dessus, le nombre de registres et leur utilisation est un paramètre important lors de la conception d'un jeu d'instructions. Par utilisation, on entend en fait le mode d'adressage.

- Adressage immédiat : la donnée est contenue dans le code opérande
- Adressage direct : le code opérande contient l'adresse d'une donnée en mémoire
- Adressage registre : le code de l'opérande contient l'adresse du registre
- Adressage indirect par mémoire : l'adresse de l'opérande (adresse mémoire) est donnée dans l'instruction
- Adressage indirect par registre : l'adresse de l'opérande est contenue dans un registre dont l'adresse est donnée par l'instruction
- Adressage de déplacement : l'adresse de l'opérande est donnée en deux parties, une base dans un registre (par exemple le PC) et un déplacement (à ajouter à une base) dans l'instruction. COMPLEXE

En général, une machine combine plusieurs types d'adressage. Le problème se pose alors lors de l'implémentation : il faut préciser le type d'adressage utilisé. Deux solutions, soit ajouter un bit qui précise le mode employé (taille d'instruction variable), soit utiliser des opcodes différents (multiplication du nombre d'opcodes).

AU niveau des jeux d'instructions on oppose traditionnellement le CISC (Complex Instruction Set Computer) ou RISC (Reduced Instruction Set Computer). Le RISC est souvent utilisé car plus simple, et caractérisé par :

- Une instruction par cycle
- Certaines instructions implantées matériellement
- Opérations register-registre
- Modes d'adressages simples et peu nombreux
- Modes d'adressages complexes réalisés à partir des simples
- Formats d'instructions simples et peu nombreux
- Taille d'instruction fixe

2. Le CPU

La CPU (Central Processing Unit) possède :

- Une ALU pour traiter les données ;
- Une unité de commande qui contrôle le mouvement des données et des instructions, ainsi que les opérations de l'ALU ;
- Des registres spécifiques pour stocker de manière temporaire les données et les instructions ;
- Un bus interne pour interconnecter les différents composants ;

Il existe deux types de registres : ceux visibles par l'utilisateur (optimisation des références à la mémoire) et ceux dits de contrôle et de statuts.

Parmi les registres visibles (utilisables via des instructions) on distingue des registres:

- De données : non utilisable pour des adresses ;
- D'adresses : souvent dévolus à un mode d'adressage particulier ;
- De conditions : suite de bits positionnés en fonction du résultat d'une opération (flag) ;
- Autres : pas de fonction spécifique ;

Parmi les registres de contrôle et de statuts, il y en a quatre suit sont primordiaux :

- Le compteur ordinal PC
- Le registre d'instruction IR
- Le registre d'adresse de mémoire MAR
- Le registre tampon mémoire MBR : fait le lien avec les registres visibles

Voici un exemple de registre de statuts : le registre PSW (program statuts word) qui contient :

- Signe
- Zéro : 1 lorsque le résultat vaut 0.
- Retenue : 1 lorsqu'il y a une retenue
- Egal : 1 lorsque le résultat d'une comparaison est une égalité
- Débordement : 1 si il y a débordement
- Interruption : indique si le fct° normal peut être interrompu
- Superviseur : indique un mode privilégié

Voici le schéma d'exécution :

- Répéter
  - Fetch : lecture de l'instruction depuis la mémoire
  - Decode : décodage de l'instruction
  - Execute : execution de l'instruction ; ceci peut être *fetch data* (lecture de données) ou *process data* (opération arithmétique ou logique) ou write data (écriture du résultat).

3. Pipeline

Le pipeline est un dispositif mis en place afin d'améliorer le temps d'exécution. L'idée est la suivante : puisqu'une instruction est composée de différentes phases, on souhaite que plusieurs phases de différentes instructions soient exécutées en même temps. En effet, puisque le fetch, le decode et l'execute sont indépendants, il semble naturel de vouloir traiter plusieurs instructions en même temps.
Ainsi, pour un cycle j d'horloge, on traite le fetch de l'instruction j, le decode de l'instruction j-1 et l'execute de l'instruction j-2. Le temps d'execution est ainsi largement diminué, puisque pour n instructions, il faut n+2 cycles d'horloge. On appelle le délai de deux cycles le temps de latence du pipeline.
Dans le cas précédent, puisque l'on est capable de gérer trois phases en même temps, on parle de pipeline à 3 étages.

Cependant, la mise en place de pipeline pause de nombreux problèmes :

- Aléa structurel : apparait lorsqu'il y a un conflit de ressources ; la solution est de suspendre l'un des pipelines concerné pendant un cycle
- Aléa de données : apparait lorsqu'un cycle dépend d'un autre cycle qui n'a pas encore été terminé (dépendance de données entre les instructions) ; la même solution que précédemment résout le problème
- Aléa de contrôle : apparait lorsque des instructions de modification de PC sont en jeu (branchement conditionnel)

VII. Complément

1. Petit mémento des registres existants

Abréviation	Terme Anglais	Terme Français
IR	Instruction Register	Registre d'Instruction
MDR	Memory Date Register	Registre de Données Mémoire
MAR	Memory Address Register	Registre d'Adresses Mémoire
PC	Program COunter	Compteur Ordinal
SP	Stack Pointer	Pointeur de Pile
ALU	Arithmetical and Logical Unit	Unité Arithmétique et Logique

2. Petit mémento des micro-opérations existantes

Abréviation	Micro-opération
Reg out	Sort le contenu du registre Reg sur le bus
Reg in	Met la valeur du bus dans le registre Reg
Lecture	Effectue une lecture de la mémoire à l'adresse MAR et met le résultat dans MDR
Ecriture	Effectue une écriture dans la mémoire à l'adresse MAR de la valeur contenue dans MDR
Attente	Permet d'attendre la fin d'une lecture ou d'écriture
ADD	Additonne les entrées A et B de l'ALU, et met le résultat sur la sortie de l'ALU F
INCRA	Incrémente l'entrée A de l'ALU (F=A+1)
DECRA	Décrémente l'entrée A de l'ALU (F=A-1)
INCRB	Incrémente l'entrée B de l'ALU (F=B+1)
DECRB	Décrémente l'entrée B de l'ALU (F=B-1)
REPA	Reporte l'entrée A de l'ALU sur sa sortie (F=A)
REPB	Reporte l'entrée B de l'ALU sur sa sortie (F=B)